オブジェクト指向プログラミング

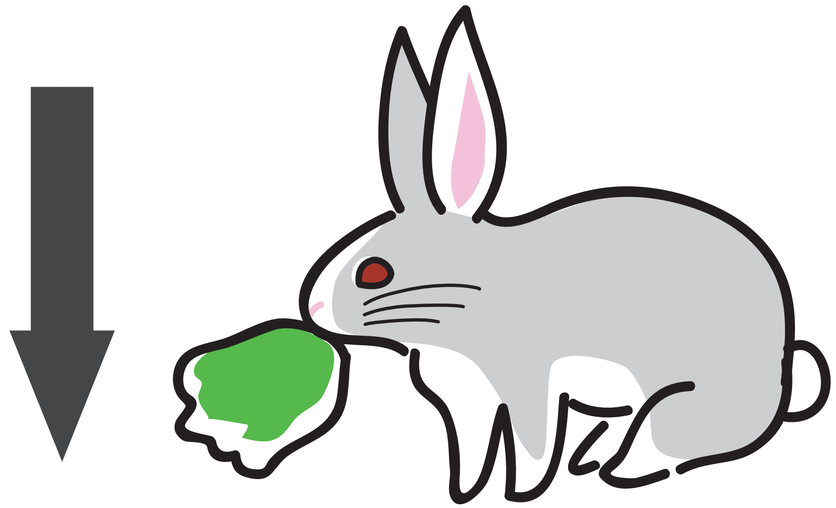
第２回

箕原辰夫

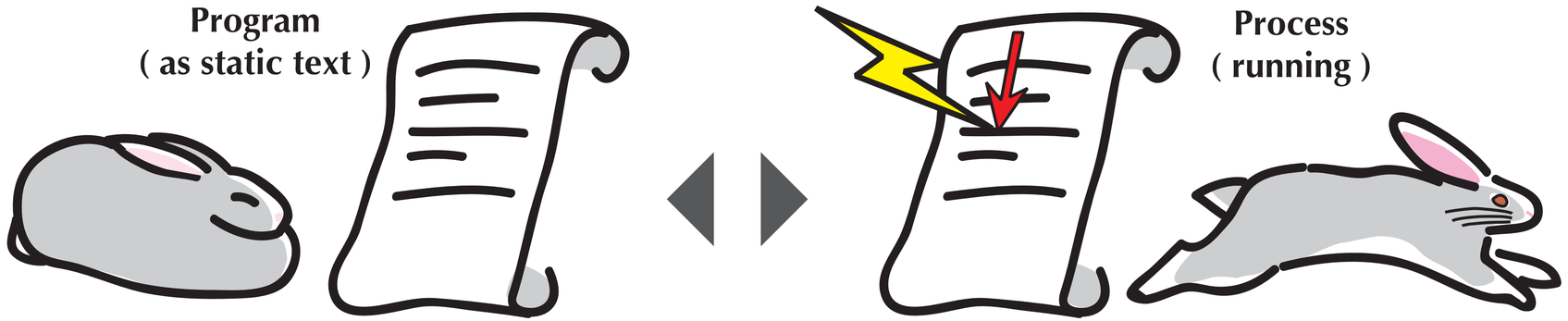
命令型プログラム

Imperative Program

Go to the center of the garden ;
Find a rabbit ;
Hold her ;
Bring her to your home ;
Give her a bit of lettuce ;
Bring back her to the garden ;



プロセスとプログラム



Pythonの文について

- １行に１つの命令を記述する
- 上から順番に実行

  例：

  ```
  print( 1 )
  print( 2 )
  print( 3 )
  ```

- １行の中で２つ以上の命令を記述する場合は、命令を

  ;（セミコロン）で区切る。 命令は、左から右に実行される

  例：print(1); print(2); print(3)

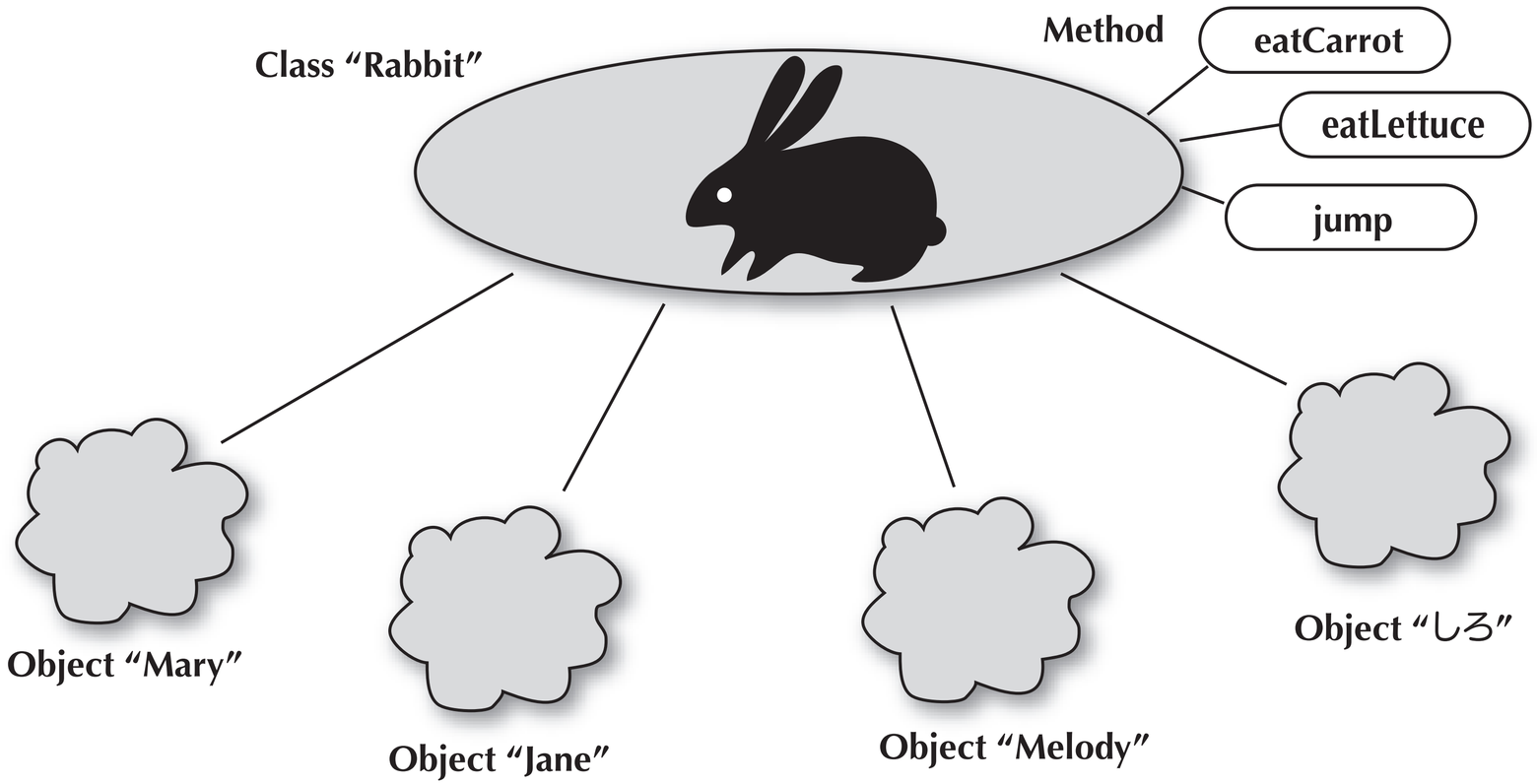
空白とコメント

- 左側に空白が空いているのがアウトラインのレベルを示すインデントと呼ばれる
  - TABキーを使う, Deleteで戻せる
  - Pythonでは、インデントがあっていないと動作しない
- コメントは、インタープリタがある程度無視してくれる注意書きのこと
  - # この後改行するまでがコメント
  - """ """ 囲まれた範囲がコメント

オブジェクト指向プログラミング言語の特徴

- 実行時にダイナミックにオブジェクトという形でメモリを獲得できる
- オブジェクトという形で保護されていて、安全にメモリにアクセスできる
- １つのオブジェクトの中に様々なプロパティ（変数や関数）をまとめることができる
- クラスを利用できるオブジェクト指向プログラミング言語は、クラスとしてオブジェクト間に共通のプロパティを記述することができる
- クラス間に継承構造がある場合は、差分的に必要な部分だけを継ぎ足すことでプログラムすることができる
- オブジェクト同士は、同時実行が可能なので、並行にプログラミングすることも可能になる

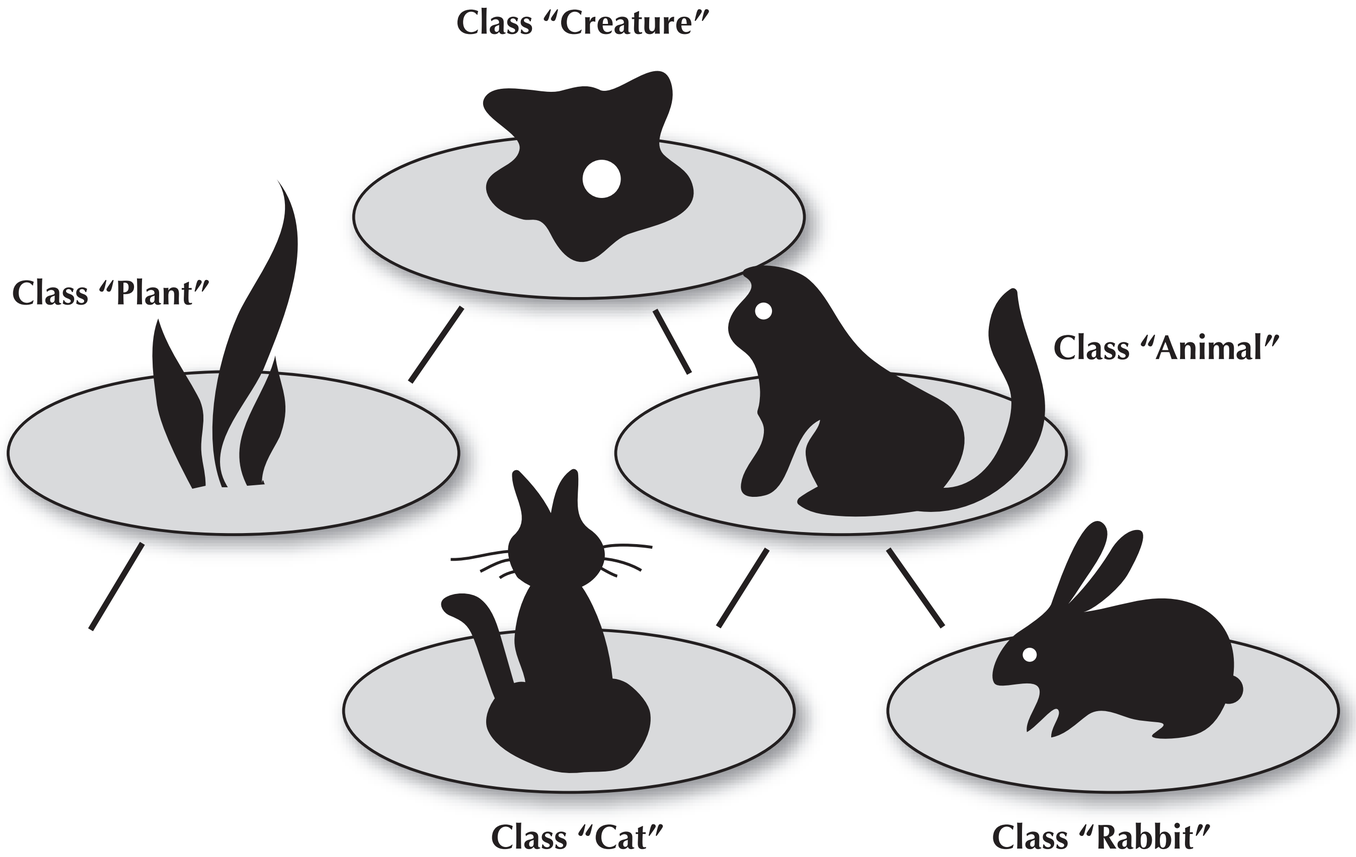
クラスとオブジェクト


Class “Rabbit”
Method
eatCarrot
eatLettuce
jump
Object “Mary”
Object “Jane”
Object “Melody”
Object “しろ”

クラスを使うのは？

- 共通の特性を持つ対象について、その特性だけをプログラムで記述すれば、同じ特性を持つ対象をどんどん増やしていける。

クラス間の階層
Inheritance

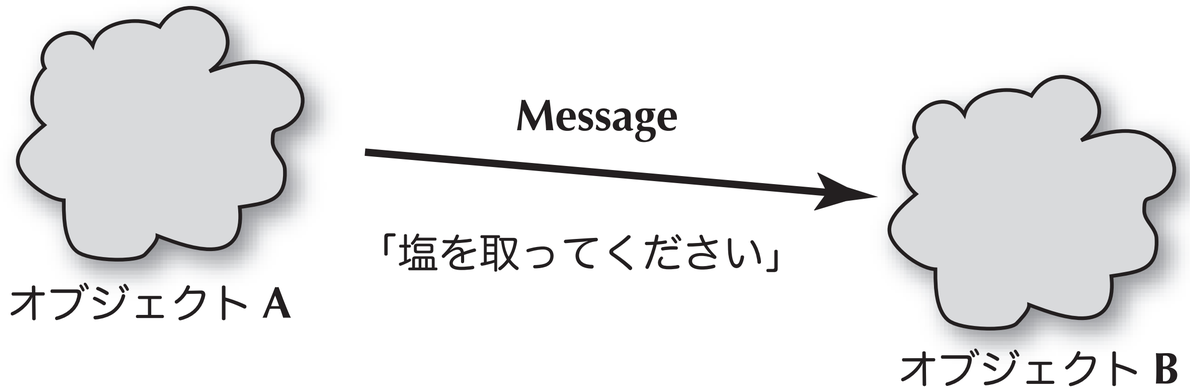


差分プログラミング

- 上位クラス（スーパークラス）で用意されている機能を使えば、その部分はプログラムする必要はない。
- 下位クラス（サブクラス）では、差分として必要な部分だけをプログラムすれば良い。
- オブジェクト指向の利点

オブジェクト指向の動作の基本（Message passing）

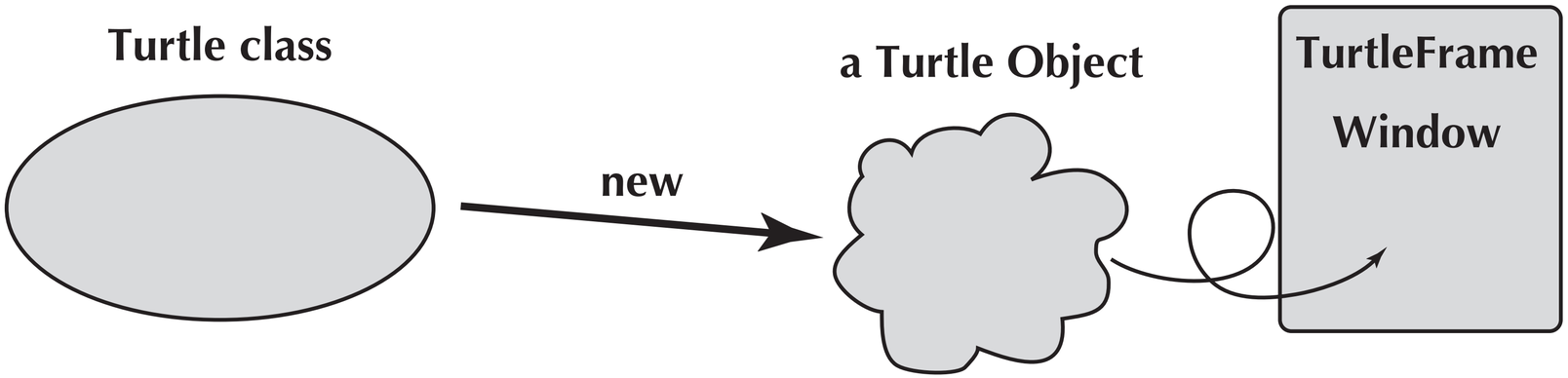
- 対象を作る
- 相手（対象：Object）を指定して、何かを頼む



オブジェクトの生成

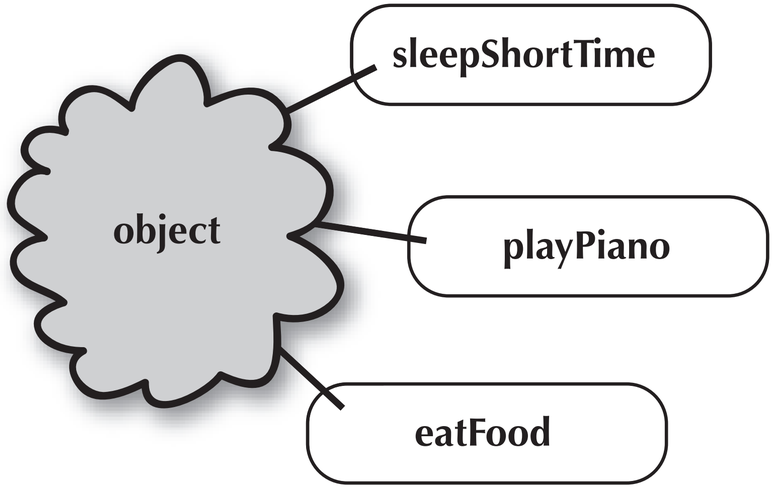
- 対象を作る書式： クラス名(パラメータ)
- Turtleクラスのオブジェクトを作る場合

➡ Turtle(window)



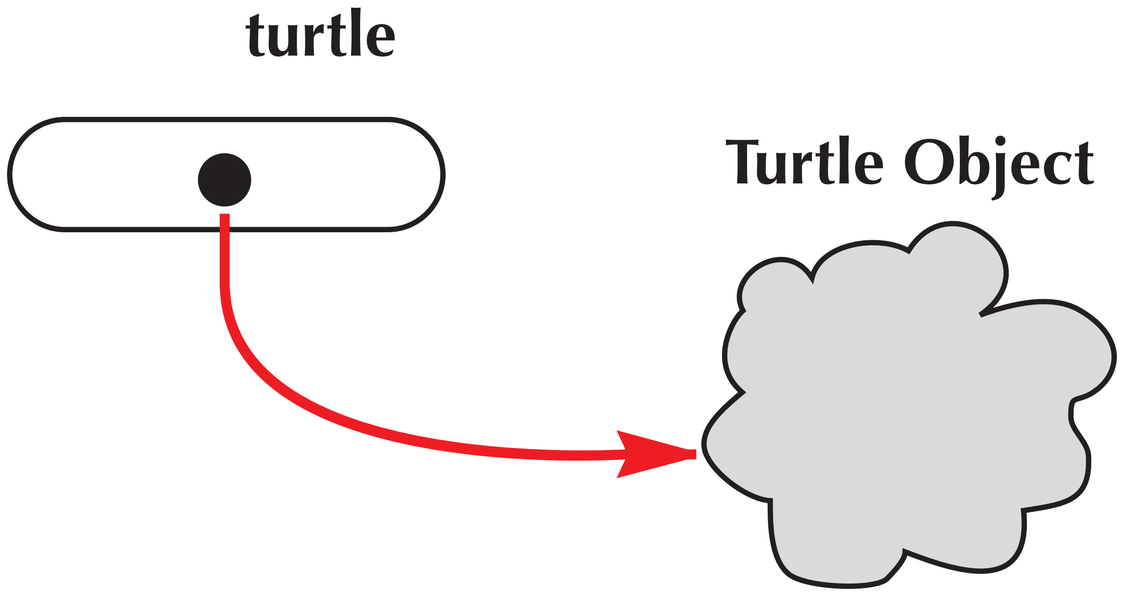
オブジェクト指向のプログラムの構造

- 相手（対象）は頼まれたことを、処理する記述がなければならない。
  - この記述としてまとめられた関数（プログラム）をメソッド（Method）と呼ぶ



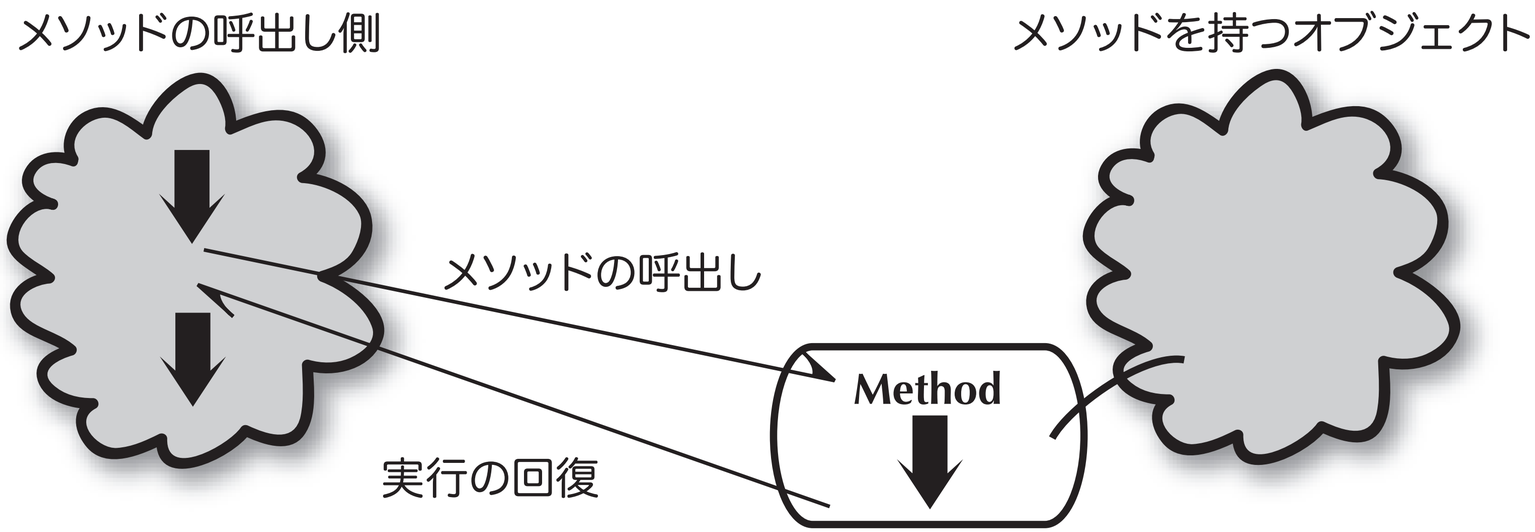
オブジェクトを保持する変数に割付け

- オブジェクト指向プログラミング言語では、変数はオブジェクトへの参照（reference）を持つ
- *turtle* = Turtle(*window*)



呼出し側と受取り側

- オブジェクト指向プログラミング言語では、相手のオブジェクトのメソッドを呼ぶ（method call）
- オブジェクト.メソッド名(パラメータ)
  - 例：*window*.drawString(“Hello”, 20, 20)



Pythonでの記述

- Pythonの文法では

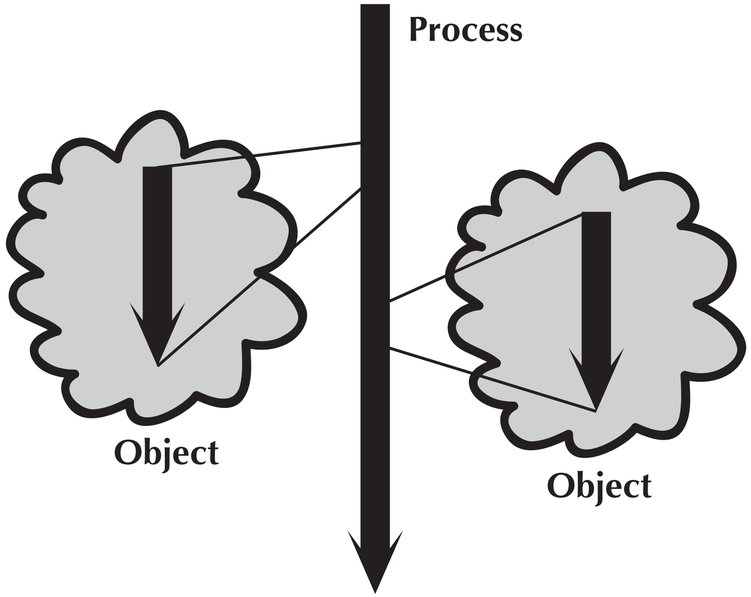
```
頼む相手.頼みたいこと( パラメータ )
```

パラメータは頼むときに渡したい情報

例：canvas.drawCircle(400, 200, 40)

オブジェクト指向の動き

- オブジェクトのメソッドを呼び出しながら、進んで行く



Pythonの記号

記号	呼び方	意味
;	セミコロン	文の区切りを示す（１行に複数の文がある場合）
.	ドット	所属を示す「〜の〜」
,	カンマ	羅列を示す「〜と〜」
" '	クォーテーション	囲んで文字列を示す
#	ナンバーマーク	行の終わりまでコメント
"""	３つの"	囲まれた範囲がコメント
*	アスタリスク	「すべて」または「掛け算」

プログラムは式を記述していく

- 式の構成要素
  - 変数
  - 定数（リテラル）
  - 演算子

- 定数は、値のこと
  - 一定の値を取り続けるということで、定数(constant value)と呼ばれる
  - プログラミング言語では、定数が型を持つ

値の型

- Java/C/C++/C#では、値の型が厳格に参照される。
  - Strict type languages　厳格な型言語
- Python/JavaScript/Rubyなどでは、型が推論されて、結果的に値の型が決まっていく
- 新規のプログラミング言語では、厳格な型言語でも型推論の機構を持っている　Swift/Rustなど
- Pythonでプログラム上で値を記述できる型（原始型：primitive type)には次のようなものがある
  - 論理型
  - 整数型
  - 実数型
  - 複素数型
  - 文字列型
- プログラム上で記述された値（型を持つ）をリテラル（literal）と呼ぶ

論理型

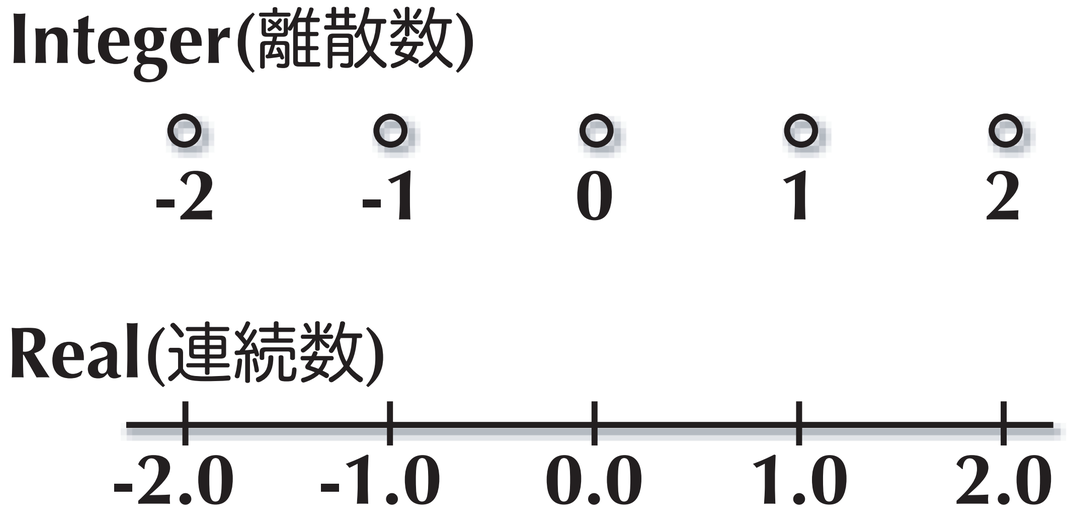
- 条件が満足されたかどうかを示す真偽値。
- **bool**型（ブール代数から）と呼ばれる。

George Boole

- 値としては、次の２つだけになっている。
  - **True**…条件を満足した（真）
  - **False**…条件を満足しない（偽）
- **None**…オブジェクトを何も指していないことを表わすが、条件式で使われた場合は、Falseとして解釈される

整数型と実数型

- 整数は、離散数（Discrete Number）と呼ばれ、実数は連続数（Linear Number）と呼ばれている。



整数型

- 整数型の型名は、int
- Pythonでは、整数の桁数の制限はない（C/C++/Java/C#などの他の言語では、32bitだと42億ぐらいまでとか、64bitだと922京ぐらいまでの制限がある）
- 長い数字列は途中で_（アンダーバー）を入れることができる（Python3.6より）
- 整数の例：
  - 7　2147483647　3
  - 79228162514264337593543950336
  - 100_000_000_000

8進数、2進数と16進数の整数

- 8進数の整数
  - 0oを先頭につける(0から7までしか使えない)
  - 0o262730

- 16進数の整数
  - 0xを先頭につける（a〜fまでが使える、大文字でも可能）
  - 0x45a

- ２進数の整数
  - 0bを先頭につける
  - 0b01010101

実数型

- 浮動小数点数で表現される
- Pythonでの型名としては、float
- 小数点をつけると自動的に実数型として認識される
  - 2.3　4. → 4.0　.05 → 0.05
- 指数表記が可能
  - 2.45×10^{16} → 2.45e16 あるいは 2.45e+16
  - 仮数×基数^{指数}　という形で標記される
- 長い数字列は途中で_（アンダーバー）を入れることができる（Python3.6より）
- 実数の例：
  - 3.14　10.　.001　1e100　3.14e-10　0e0
  - 3.14_15_92_65_35_89_79

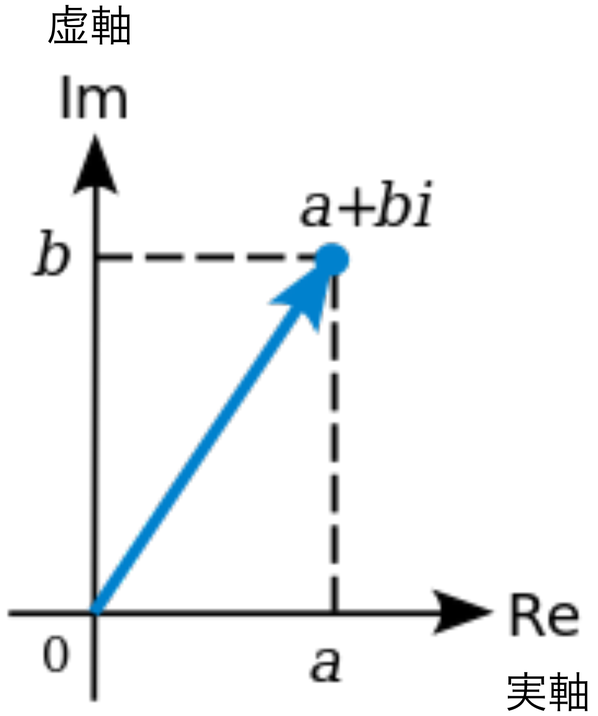
複素数型

- 虚数（数学ではiを使うが、工学系やPythonではjを用いる）

$$i \times i = -1 \qquad \sqrt{-1} = i$$

- 虚数値の例：
  - 1j　3.14j　10.j　10j　.001j　1e100j
  - 3.14e-10j　3.14_15_93j

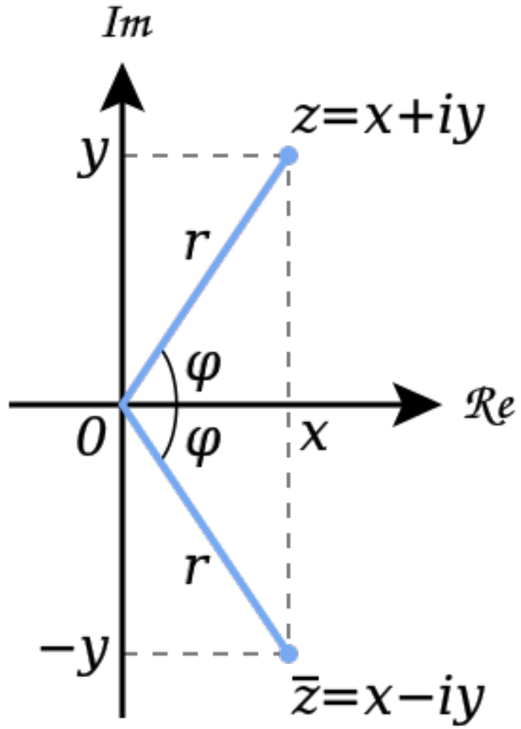
- 実数部と虚数部を加減算することで
  複素数を表現することができる
- 型名は、complex
- 複素数の値の例：
  - 12+4j　3.5-5.2j　2+5j



複素数平面（Complex plane）

- 複素数平面は、ガウス平面（Gaussian plane）とも呼ばれる
- 共役複素数は、conjugateを使って求めることができる
  - 例：(1+1j).conjugate( )

- 実数部を取り出すのは、.real
  虚数部を取り出すのは、.imag
  を用いる
  - 例：(1+0.5j).real⇒1.0
    (3+4j).imag⇒4.0



複素数とオブジェクト

- 複素数はオブジェクトになっている（実は整数や実数もオブジェクト）
- オブジェクトは、そのオブジェクト固有の属性（attribute, フィールドfield, インスタンス変数instance variableとも呼ぶ）を持っており、通常の変数としてアクセスできる
- オブジェクトは、そのオブジェクト固有の関数（function、メソッドmethodと呼ばれる）を持っており、そのオブジェクトを介して呼び出すことができる
- 属性とメソッドを合わせて、プロパティ（property）と呼ぶことがある

Pythonでのオブジェクトのプロパティの表記法

- 属性を参照する場合
  - オブジェクト.属性名
  - 例：(3+4j).real…3.0が返される
  - 例：(3+4j).imag…4.0が返される
- メソッドを呼び出す場合
  - オブジェクト.メソッド名(パラメータ)
  - パラメータが必要ないときは括弧内には何も書かない
  - 例：(3+4j).conjugate( )…3-4jが返される

文字列型

- 文字はすべてUnicodeで符号化されている。
- ファイルなどを読み込むときに、別の符号（JISやShift JIS）を用いるときは、符号（エンコーディング：encoding）の指定をしなければならない。

- 文字列はstrクラスになっている(str関数)　※ドキュメント上はstring
- Pythonでは、どちらかの引用符を用いる
  - 値を記述するときは一重引用符で囲む 'a'
  - 値を記述するときは二重引用符で囲む "a"

JIS/Shift JIS/EUC/Unicodeの違い

- ASCII…アメリカが作った英語用のコード（１バイト）0x00～0x7f
- JIS X0208…日本でつくられた２バイトの漢字・かなどが入った文字コード体系
- JIS X0201…日本でつくられた１バイトの半角カナを含む文字コード0x80以降も含む　　\…ASCIIでの0x5c　￥…JIS X0201での0x5c
  - C:\Program Files\Excel.exe　←　C:￥Program Files￥Excel.exe
- JIS X0208の漢字コードと半角カナを両方使いたい→Shift JISコード
- EUCは、１バイト単位でJIS＋0x80
- Unicodeは、Apple/Microsoftなどアメリカが勝手に考えたコード

Unicodeによる文字列

- stringクラスという形で定義されている
- \nは改行、\tは水平タブ
- \" はダブルクォーテーション、\'はシングル
- \\ はバックスラッシュ自身を表わす
- \uを用いて16進数4桁で、文字を指定できる。
  - \u4e00 → 一
- \Uで拡張部分のUnicodeを指定できる（16進数8桁）

- パレットでUnicodeのコード表を見てみましょう（以下Mac OS X）
  - 環境設定（キーボード）→キーボードタブ
    →「メニューバーに絵文字とキーボードビューア」を表示にチェック
  - 言語（スクリプト）メニューから、「絵文字と記号を表示」→歯車メニューから「リストをカスタマイズ」→「Unicode」にチェックを入れる

- IMEパッドでUnicodeのコード表を見てみましょう（以下Windows）
  - Windows IMEパッドを、タスクバーのIMEのアイコン（「あ」とか「A」とか出ているもの）を右クリックで出るメニューから開く
  - 文字カテゴリでUnicodeを選ぶ
  - 基本多言語面は、4桁で指定する部分
  - 追加多言語面は、8桁で指定する部分

Unicodeと組込み関数

- 1文字に対して
  - ord(1文字の文字列)…文字に対応するUnicodeのコードを整数として返してくれる
    - 例：ord("🙇")→77848　　ord("A")→65　　ord("漢")→28450
  - chr(整数)…Unicodeの整数に対応する1文字の文字列を返してくれる
    - 例：chr(105)→'i'　　chr(0x3fe9)→'𰾩'　　chr(0x103b5)→'𐎵'
- 文字列に対して
  - ascii(文字列)…ascii文字はそのまま、それ以外の文字については、文字列中の各文字に対して対応するUnicodeの16進数を「文字列リテラル」として返してくれる。もちろん、 1文字の文字列でも使用可能
    - 例：ascii("ABCあい🙇")→'ABC\\u3042\\u3044\\U00013018'
    - 例：ascii("鮪") → '\\u9baa'　　ascii("𓀉") → '\\U00013009'

構造的な型のリテラル

- タプル
  - 書式：(値, 値, …)
  - ()は、空のタプル
  - (12,)は、要素が1つのタプル
  - 複数の値をまとめることができる
  - それぞれの要素を書き換えることはできない
  - 例：　(34, "ABC", 78.9)
  - (23.4, 45.6, 56.7)
- リスト
  - 書式：[値, 値, …]
  - []は、空のリスト
  - [23]は、要素が1つのリスト
  - 複数の値をまとめることができる
  - それぞれの要素を取り換えることが可能
  - 例：　[12, 23, 34, 45]
  - ["たらこ", "いくら", "うに"]
  - [34, "ABC", 78.9]

構造的なリテラル（その他）

- 集合
  - 書式：{ 値, 値, … }
  - {}は、空の（要素のない）集合または辞書
  - { 12 }は、要素が1つの集合
  - 複数の値をまとめることができる
  - 同じ値の要素は 1 つしか登録されない
  - 順番は問われない（インデックス式・スライス式は使えない）
  - 例：{"うめ", "もも", "さくら" }
  - { 34, "ABC", 78.9, True }
- 辞書
  - 書式：{ キー：値, キー：値, … }
  - { "one":1 }は、要素が1つの辞書
  - キーと値の組合わせを複数まとめることができる
  - 同じキーの要素は 1 つしか登録されない
  - 順番は問われない（スライス式は使えない）
  - 例：　{ "one" : 1, "two" : 2 }
  - { 34: 56, "ABC": 12, 78.9: "A" }